INTERNATIONAL STANDARD

IEC 62258-2

First edition
2005-06

Semiconductor die products –

Part 2:
Exchange data formats

Reference number
IEC 62258-2:2005(E)

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site (www.iec.ch)**

- **Catalogue of IEC publications**

  The on-line catalogue on the IEC web site (www.iec.ch/searchpub) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.

- **IEC Just Published**

  This summary of recently issued publications (www.iec.ch/online_news/ justpub) is also available by email. Please contact the Customer Service Centre (see below) for further information.

- **Customer Service Centre**

  If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

  Email: custserv@iec.ch
  Tel: +41 22 919 02 11
  Fax: +41 22 919 03 00

INTERNATIONAL STANDARD

IEC 62258-2

First edition
2005-06

Semiconductor die products –

Part 2:
Exchange data formats

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE XB

For price, see current catalogue

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

SEMICONDUCTOR DIE PRODUCTS –

Part 2: Exchange data formats

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 62258-2 has been prepared by IEC technical committee 47: Semiconductor devices.

The text of this standard is based on the following documents:

FDIS	Report on voting
47/1809/FDIS	47/1822/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This Part of IEC 62258 should be read in conjunction with IEC 62258-1.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

IEC 62258, as currently conceived, consists of the following parts, under the general title *Semiconductor die products*

Part 1: Requirements for procurement and use

Part 2: Exchange data formats

Part 3: Recommendations for good practice in handling, packing and storage

Part 4: Questionnaire for die users and suppliers

Part 5: Requirements for information concerning electrical simulation

Part 6: Requirements for information concerning thermal simulation

Further parts may be added as required.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

INTRODUCTION

This International Standard is based on the work carried out in the ESPRIT 4th Framework project GOOD-DIE which resulted in the publication of the ES 59008 series of European specifications. Organisations that helped prepare this standard included the ESPRIT GOOD-DIE and ENCAST projects, the Die Products Consortium, and JEITA.

SEMICONDUCTOR DIE PRODUCTS –

Part 2: Exchange data formats

1 Scope

This part of IEC 62258 has been developed to facilitate the production, supply and use of semiconductor die products, including but not limited to

– wafers,
– singulated bare die,
– die and wafers with attached connection structures,
– minimally or partially encapsulated die and wafers.

This standard specifies the data formats that may be used for the exchange of data covered by other parts in the IEC 62258 series as well as definitions of all parameters used according to the principles and methods of IEC 61360-1, IEC 61360-2 and IEC 61360-4. It introduces a Device Data Exchange (DDX) format, with the prime goal of facilitating the transfer of adequate geometric data between the die manufacturer and the CAD/CAE user and formal information models that allow data exchange in other formats such as STEP physical file format, fin accordance with ISO 10303-21 and XML. The data format has been kept intentionally flexible to permit usage beyond this initial scope.

This standard reflects the DDX data format: version 1.2.1.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 62258-1, *Semiconductor die products – Part 1: Requirements for procurement and use*[1]

IEC 61360-1:2002, *Standard data element types with associated classification scheme for electric components – Part 1: Definitions – Principles and methods*

IEC 61360-2:2002, *Standard data element types with associated classification scheme for electric components – Part 2: EXPRESS dictionary schema*

IEC 61360-4:1997, *Standard data element types with associated classification scheme for electric components – Part 4: IEC reference collection of standard data element types, component classes and terms*

ISO 6093:1985, *Information processing – Representation of numerical values in character strings for information interchange*

ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times*

ISO 10303-21:2002, *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*

[1] To be published.

3 Terms and definitions

For the purposes of this document, the definitions as given in IEC 62258-1 shall apply.

4 Requirements

Specific reference for Parameter Variables is made to the IEC 61360 Data Element Type (DET) codes, which are defined in Part 4 of IEC 61360-4.

5 Device Data eXchange format (DDX) file goals and usage

To facilitate the transferral of data by electronic media from the device vendor to the end-user for use within a CAD or CAE system, a data file format, **D**evice **D**ata e**X**change, (**DDX**), shall be used. This data file format has been deliberately kept flexible, to permit further enhancements and additions for future use.

It is strongly recommended that **D**evice **D**ata e**X**change files have the three letter **DDX** file extension, and a **D**evice **D**ata e**X**change file shall hereon be referred to as a **DDX** file.

5.1 Data that are to be transferred from a device vendor to a user shall be contained in a single computer-readable DDX file, and the minimum contents of this file shall suffice a geometric CAD/CAE software design system. The file shall be textually readable, to permit simple manual verification.

5.2 The DDX file and its data contents shall be independent of both computer machine and operating system.

5.3 The DDX file contents shall include mechanical and interconnectivity information, but may additionally include electrical and functional data.

5.4 The DDX file may contain data for one or more devices and shall be capable of being used as a library file by a CAD/CAE software design system. The file may contain one or more sets of data for the same device type, each having different delivery forms, such as bumped die, bare die, and Chip-Scale packaging.

5.5 The DDX file shall be capable of being simply or automatically generated, such as by an ASCII text editor or a spreadsheet.

5.6 The DDX file shall be capable of referencing additional external files, such as simulation and thermal model files.

5.7 All data shall be defined in such a way that conversion to or from other exchange formats is possible, such as GDSII and CIF for geometric data of die. As close a compatibility to the existing DIE (Die Information Exchange) data as possible is desired, to facilitate simple translation of partial DIE data files.

5.8 Definitions of parameters shall be in conformity with IEC 61360-1 (refer to Clause 5 of IEC 62258-1).

6 DDX file format and file format rules

NOTE Version 1.2.1 of DDX supersedes version 1.0.0 contained in ES 59008-6-1 [1][2].

The **DDX** file shall be an ASCII compatible text file with suitable line termination. Line termination will depend upon the operating system. DOS/Windows® generally uses a carriage/line-feed <CR/LF> terminator (ASCII 0Dh/0Ah), whereas UNIX® invariably relies solely upon a line-feed <LF> (ASCII 0Ah) terminator, the carriage return <CR> (ASCII 0Dh) being present by implication.

6.1 All data not complying to the data syntax (refer to 7.3) shall be treated as a remark and, as such, ignored.

6.2 All mandatory data shall be present. Missing data shall be flagged as an error, rendering that data unusable.

6.3 ASCII characters 00h to 7Fh are permitted, ASCII characters 80h to FFh shall be ignored.

6.4 All text data shall be case independent.

6.5 Underscores "_" shall be ignored in a variable or property name, and may be used as intermediate name separators. Underscores are valid within textual string and name data.

6.6 A comma "," shall be used as a data separator.

6.7 All data lines shall be terminated with a semicolon, ";".

6.8 Braces are used to open and close structures or BLOCKs. An open brace "{" shall be used to begin a structure or block, and a close brace "}" shall be used to terminate a structure or block.

6.9 Brackets "()" shall be permitted, then ignored, in numeric data for clarity (e.g. in coordinate pairs).

6.10 To accommodate typical spreadsheet CSV (Comma Separated Variable) format outputs, textual data may be inside double quotes "", and matching pairs of double quotes shall be ignored.

6.11 Mathematical operations, calculations or formulae shall not be permitted within numeric data.

6.12 Space characters (ASCII 20h) and tab characters (ASCII 09h) shall both be treated as space separators, multiple space and tab characters will syntactically be treated as a single space separator.

6.13 Lines beginning with a hash "#" shall be treated as an intentional comment. All data on that line shall be ignored.

7 DDX file content

7.1 DDX file content rules

7.1.1 Block structure

Data shall only exist within a block structure, referred to as a DEVICE block, and one or more DEVICE blocks, each containing data, may exist within a single file. Each DEVICE block is unique, and shall only contain data relevant to a single device, having a specific device form. All data within each DEVICE block shall be treated as being local and unique only to that block. (Refer to 6.8)

---

2 Figures in brackets refer to the bibliography.

7.1.2 Parameter types

There are two types of parameters use for data, structures and variables, and these parameters shall only exist with a DEVICE block:

– A structure determines a set or multiple sets of data having different data types.
– A variable is equated to a single or multiple data of a single data type.

7.1.3 Data types

Data types comprise:

7.1.3.1 Textual string data

All ASCII characters from ASCII 20h to ASCII 7Fh are permitted within textual data, characters including and above ASCII 80h shall be ignored. Consideration may be given to special print and display control characters to permit the printing of underscore or overscore characters. It is advised that textual string data are placed within pairs of double quotes, refer 6.10.

7.1.3.2 Textual name data

All names shall be unique, and shall only consist of the following characters from the ASCII character set:-

A-Z a-z 0-9 $ – % & ! @ _ .

When textual name data are used to form a file name, it is advisable for the name to be limited to eight characters for the file name and to three characters for the file extension, with a point "**.**" used as the name/extension delimiter, in line with many common operating systems. It is advisable for textual name data to be placed within pairs of double quotes (refer to Clause 6).

Note that all textual data are case independent, and spaces are not permitted within a textual name.

7.1.3.3 Real numeric data

Real numeric data shall comply to ISO 6093:1985, and shall consist of the following characters:

0-9 + – . E e

The data values may be signed, and use engineering or scientific notation, but shall not include dimensional units, e.g.

90008, 9000.80, 9.0008E5, -5207, -5.207E3, 0.102, 102E-3

Note that a comma "**,**" is used as a data separator, and therefore shall not be used as a replacement for a decimal point "**.**".

7.1.3.4 Integer numeric data

Integer numeric data values shall comply with ISO 6093:1985, and only the characters **0** to **9** are permitted. Integers shall be unsigned, and shall not include dimensional units.

For practical purposes, an integer shall be limited to 16-bit resolution, i.e. integer values between and including 0 to 65536 only are acceptable.

7.1.3.5 Date data

Date data values shall comply with ISO 8601:2000 format, Yr2000 compliant, and may include time information as well e.g.

"YYYY-MM-DD", "YYYYMMDD", "YYYY-MM-DDTHH:MM:SS".

7.1.4 Forward references

To permit single-pass parsing, no variable identifier or variable name shall be referenced prior to being defined.

7.1.5 Units

All units shall belong to the SI system, apart from the geometric unit of the micron (10^{-6}m), the inch and the mil (10^{-3} inch). Only one unit of dimension shall be permitted within a single **DEVICE** block. Note that the inch and the mil are non-preferred units, and are only present due to continued common usage.

7.1.6 Coordinate data

In all coordinate data, the **X** coordinate shall precede the **Y** coordinate and the **Y** coordinate shall precede the **Z** coordinate (i.e. **X,Y** or **X,Y,Z**).

The **X** coordinate shall be the horizontal axis (numerically left to right), the **Y** coordinate shall be the vertical axis (numerically bottom to top), and the **Z** coordinate shall be depth axis (numerically near to far).

7.2 DDX DEVICE block syntax

DEVICE device_name device_form {
relevant die data ……
}

The **DDX** file may contain one or more **DEVICE** blocks, all data pertaining to a particular device shall be embedded within the relevant block. (refer to 6.1 and 7.1.1).

A **DEVICE** block is opened by the **DEVICE** keyword and opening brace "**{**", (as shown), and the **DEVICE** block is closed by the matching closing brace "**}**".

Data not within a **DEVICE** block structure shall be treated as a remark, permitting the future addition of checksum information, file creation date and historical data etc., within the **DDX** file, without affecting the actual device data.

The **device_name** is the given name by which the device shall be referred, and the **device_form** is the mechanical form of the device to which the block data pertains.

Valid data for the **device_form** variable are

- **bare_die,**
- **bumped_die,**
- **lead_frame_die**
- **minimally_packaged_device (or MPD).**

Further **device_form** types may be added at a later stage, refer to IEC 61360-4, AAD004-001, "die type code", for further details.

Only one **DEVICE** block having **device_name** of type **device_form** shall be present within the **DDX** file, but duplication of either **device_name** or **device_form** is permissible.

An example of a typical **DDX** file arrangement of **DEVICE** blocks is as follows:

```
DEVICE name1 bare_die {
relevant data for device "name1" as a bare die..
}
DEVICE name1 bumped_die {
relevant data for device "name1"as a bumped die..
}
DEVICE name2 mpd {
relevant data for device "name2" as a minimally packaged device..
}
DEVICE name2 bare_die {
relevant data for device "name2" as a bare die..
}
DEVICE name1 mpd {
relevant data for device "name1" as a minimally packaged device..
}
DEVICE name3 bare_die {
relevant data for device "name3" as a bare die..
}
```

In the above example, there are three occurrences of a **DEVICE** block for device "name1", and two occurrences of a **DEVICE** block for device "name2", but each of these **DEVICE** blocks specify a different **device_form**. The order or sequencing of the **DEVICE** blocks has no relevance.

7.3 DDX data syntax

Property = value [, value];

<**property**>[*equate separator*]<value/variable {*separator* <value/variable> }>[*data terminator*][*line terminator*]

<**property**>	::=	Parameter name
[*space*]	::=	{space character (20h) or tab character (09h)}0+
[*equate separator*]	::=	[space]{equal **=**}[space]
[*separator*]	::=	[space]{comma **,**}[space]
[*data terminator*]	::=	[space]{semicolon;}
[*line terminator*]	::=	{CR or CR/LF}

For example:

```
thickness   =   100.0 ;
Thickness=470;
geometric_units=micron;
geometricunits     =     micron;
GeometricUnits= "millimetres";
terminal_type = T1, Circle, 220;
Terminal_Type = T2, Rectangle,  200 ,  250;
TerminalType = T2, O, (200, 250);
TERMINALTYPE     =     T2, O,  200  ,  250;
```

Thus, **terminal_type, Terminal_Type**, **TerminalType** and **TERMINALTYPE** will all reference the same parameter name.

8 Definitions of DEVICE block parameters

Where a parameter is unique to the **device_form**, as defined in the **DEVICE** block, the parameter will be preceded with the following:

DIE_	data parameter is unique to bare die or bumped die form
BUMP_	data parameter is unique to only bumped die
MPD_	data parameter is unique to a minimally packaged device, such as a CSP
WAFER_	data parameter is unique to a die device delivered at wafer level
LEAD_	data parameter is unique to a die device with attached lead frame.

Within the following list of data parameters (8.1 onwards), the following are shown:

– the parameter name, as used syntactically within the **DDX** file,
– the parameter type, indicating either a variable or structure data type,
– the parameter function, determining its usage and meaning,
– the parameter value, indicating the type of data expected,
– any parameter limitation, indicating any limitation within the **DEVICE** block,
– parameter dependencies, highlighting parameters that need to be declared prior to invocation,
– one or more practical examples, and
– any relevant notes.

A brief table of parameters is given in Annex F, and a working example of a full **DDX DEVICE** block is given in Annex A, with its expected graphical output in Annex C.

All parameters shall conform to the relevant IEC 61360 Data Element Type (DET) codes, as defined in IEC 61360-4.

Terms and conventions

A point of electrical connection is called a terminal. This may be a bond-pad for a bare die, and may equally refer to the landing or connection footprint area required by an interconnection medium. It is a common convention for die to have the initial terminal, numbered 1, in the upper left hand corner of the die, and for terminal or pin numbering to continue counter-clockwise in sequence.

The X coordinate dimensions are for the length in the horizontal plane with increasing positive values to the right. The Y coordinate dimensions are for the width in the horizontal plane with increasing positive values away from the user's view. The Z coordinate dimensions are for height in the vertical direction with increasing positive values upwards.

As a point of reference, all die components, including bumped die, are generally viewed from above with the active side upwards.

Summary of general rules

- All valid data shall be contained within a **DEVICE** block (refer to 7.1.1).
- Any local or unique parameter, such as a name, shall be defined prior to its usage.
- All parameters shall conform to the relevant IEC 61360 DET codes, as defined in IEC 61360-4 (refer to 5.8).
- The units of measurement, **GEOMETRIC_UNITS**, shall be defined before any geometric variable is defined.
- The geometric origin, **GEOMETRIC_ORIGIN**, and the geometric view, **GEOMETRIC_VIEW**, shall be defined before any geometric coordinates are defined.

- The **TERMINAL_COUNT** parameter shall be defined before any **TERMINAL** parameters are referred to and the number of **TERMINAL** parameters shall not exceed the **TERMINAL_COUNT** value.
- The **TERMINAL_TYPE_COUNT** parameter shall be defined before any **TERMINAL_TYPE** parameters are referred to and the number of **TERMINAL_TYPE** parameters shall not exceed the **TERMINAL_TYPE_COUNT** value.

8.1 BLOCK_CREATION_DATE Parameter

Parameter Name	**BLOCK_CREATION_DATE**
Parameter Type	Variable
Parameter Function	Specifies the date that the **DEVICE** block was created and last edited.
Parameter Values	Date, as described in 7.1.3.5
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`BLOCK_CREATION_DATE = “1997-12-25”;`
Notes	Refer to Annex G

8.2 BLOCK_VERSION Parameter

Parameter Name	**BLOCK_VERSION**
Parameter Type	Variable
Parameter Function	Specifies the version number and/or issue number of the **DEVICE** block.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`BLOCK_VERSION = “1.0A”;`
Notes	Refer to Annex G

8.3 VERSION Parameter

Parameter Name	**VERSION**
Parameter Type	Variable
Parameter Function	Specifies the revision/version number of the DDX standard, (currently at version 1.2.1), to which this **DEVICE** block conforms.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`VERSION = “1.2.1”;`
Notes	Refer to Annex G

8.4 DEVICE_FORM Parameter

This is defined within the **DEVICE** block heading by the **device_form** parameter.

Parameter Name	**device_form**
Parameter Type	Variable, refer to 7.2 on **DEVICE** blocks
Parameter Function	Defines the physical form of the device.
Parameter Values	Textual string data, as described in 7.1.3.1
Example	`bare_die, bumped_die, MPD`
Notes	Refer to 7.2 and Annex G.

8.5 DEVICE_NAME Parameter

This is defined within the **DEVICE** block heading as the **device_name** parameter

Parameter Name	**device_name**
Parameter Type	Variable, refer to 7.2 on **DEVICE** blocks
Parameter Function	Defines the device’s reference name.
Parameter Values	Textual name data, as described in 7.1.3.2
Example	`SN74LS04`
Notes	Refer to 7.1.6, 7.2 and Annex G.

8.6 DIE_MASK_REVISION Parameter

Parameter Name	**DIE_MASK_REVISION**
Parameter Type	Variable
Parameter Function	Specifies the mask revision details associated with that particular version on the die.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`DIE_MASK_REVISION = "RTDAC1"` `DIE_MASK_REVISION = "9033-2-101-M5/2"`
Notes	Intended to ensure that the die data matches the geometric version of the actual die. Refer to Annex G.

8.7 MANUFACTURER Parameter

Parameter Name	**MANUFACTURER**
Parameter Type	Variable
Parameter Function	Specifies the manufacturer or fabrication house of the device.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`MANUFACTURER = "Fuzziwuz Logic Inc.";`

8.8 DIE_NAME Parameter

Parameter Name	**DIE_NAME**
Parameter Type	Variable
Parameter Function	Specifies the name of the die or mask set from which the die was produced.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`DIE_NAME = "XXC345";`
Notes	Refer to Annex G.

8.9 DIE_PACKAGED_PART_NAME Parameter

Parameter Name	**DIE_PACKAGED_PART_NAME**
Parameter Type	Variable
Parameter Function	Specifies the manufacturers part name for the equivalent packaged part, where available or applicable.
Parameter Values	Textual string data, as described in 7.1.3.1
Example	`DIE_PACKAGED_PART_NAME = "SN5405JN";`
Notes	Used to reference the identical packaged die part, not merely similar function, when supplied by the same manufacturer in packaged form.

8.10 FUNCTION Parameter

Parameter Name	**FUNCTION**
Parameter Type	Variable
Parameter Function	A brief description of the devices' function.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`FUNCTION = "16 Bit Microprocessor";`
Notes	IEC 61360-4 specifies certain functional and application classes that may be used.

8.11 DATA_SOURCE Parameter

Parameter Name	**DATA_SOURCE**
Parameter Type	Variable
Parameter Function	Specifies the source of the device data, essentially where this is different from the manufacturer or fabrication house.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`DATA_SOURCE = “AnyChip Technology Ltd.”;` `DATA_SOURCE = “Good-Die database”;`

8.12 DATA_VERSION Parameter

Parameter Name	**DATA_VERSION**
Parameter Type	Variable
Parameter Function	Specifies the revision of the data source use to determine the parameters within the **DEVICE** block. This parameter is linked to **8.11 DATA_SOURCE** parameter.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`DATA_VERSION = “Initial Issue 1.0”;`
Notes	Refer to Annex G

8.13 GEOMETRIC_UNITS Parameter

Parameter Name	**GEOMETRIC_UNITS**
Parameter Type	Variable
Parameter Function	Specifies the geometric units that shall apply to all geometric values within the **DEVICE** block.
Parameter Values	Textual string data, as described in 7.1.3.1. One of the following values: ▪ micrometre, or micron, ▪ metre, ▪ millimetre, ▪ inch ▪ mil (1.0E-3 inch)
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`GEOMETRIC_UNITS = microns;` `GEOMETRIC_UNITS = mil;`
Notes	The **GEOMETRIC_UNITS** parameter shall be declared before any geometric units are used. The micron is generally the default dimensional unit for die dimensions, and the inch and mil are non-preferred units. Refer to 7.1.5 and Annex E.

8.14 GEOMETRIC_VIEW Parameter

Parameter Name	**GEOMETRIC_VIEW**
Parameter Type	Variable
Parameter Function	Specifies the geometric view that shall apply to all geometric shapes within the **DEVICE** block.
Parameter Values	Textual string data, as described in 7.1.3.1. One of the following values: • **TOP** meaning active side upwards, and • **BOTTOM** meaning active side downwards.
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`GEOMETRIC_VIEW = top;` `GEOMETRIC_VIEW = bottom;`
Notes	The **GEOMETRIC_VIEW** parameter shall be declared before any geometric shapes are created. It would be common for a bare die and packaged part to be viewed in the “TOP” view, whereas bumped die may well be viewed from the “BOTTOM” (i.e. through the substrate). Refer to Annex E.

8.15 SIZE Parameter

Parameter Name	**SIZE**
Parameter Type	Variable
Parameter Function	Determines the X- and Y- dimensions of the device, and optionally specifies its shape as an ellipse.
Parameter Values	Real X-dimension, Real Y-dimension, (as described in 7.1.3.3), {Ellipse}
Dependencies	**GEOMETRIC_UNITS**, **GEOMETRIC_VIEW**
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`SIZE = 250, 500.5;` `SIZE = 350, 350, E;`
Notes	Dimension values are in **GEOMETRIC_UNITS**. In the latter example, the character "E" as the 3rd parameter defines an ellipse, in this instance a circular die of 350 **GEOMETRIC_UNITS** in diameter. Refer to Annex E.

8.16 THICKNESS Parameter

Parameter Name	**THICKNESS**
Parameter Type	Variable
Parameter Function	Determines the thickness (Z-dimension) of the device.
Parameter Values	Real Z-dimension value, as described in 7.1.3.3
Dependencies	**GEOMETRIC_UNITS**
Limitations	Shall be declared only once within a single DEVICE block.
Example	`THICKNESS = 10.5;`
Notes	Dimension values are in **GEOMETRIC_UNITS**.

8.17 GEOMETRIC_ORIGIN Parameter

Parameter Name	**GEOMETRIC_ORIGIN**
Parameter Type	Variable
Parameter Function	Determines the X- and Y-geometric origin from which all other coordinate pairs are referenced. The origin is given with respect to the geometric centre of the die in the units specified by the **GEOMETRIC_UNITS** parameter.
Parameter Values	Real X coordinate origin, Real Y coordinate origin, as described in 7.1.3.3
Dependencies	**GEOMETRIC_UNITS**, **SIZE**
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`GEOMETRIC_ORIGIN = -6000,-7500;`
Notes	Dimension values are in **GEOMETRIC_UNITS**. The origin coordinate pair values relate to the geometric die centre, refer to Figure 1. for further explanation, and Annex E.

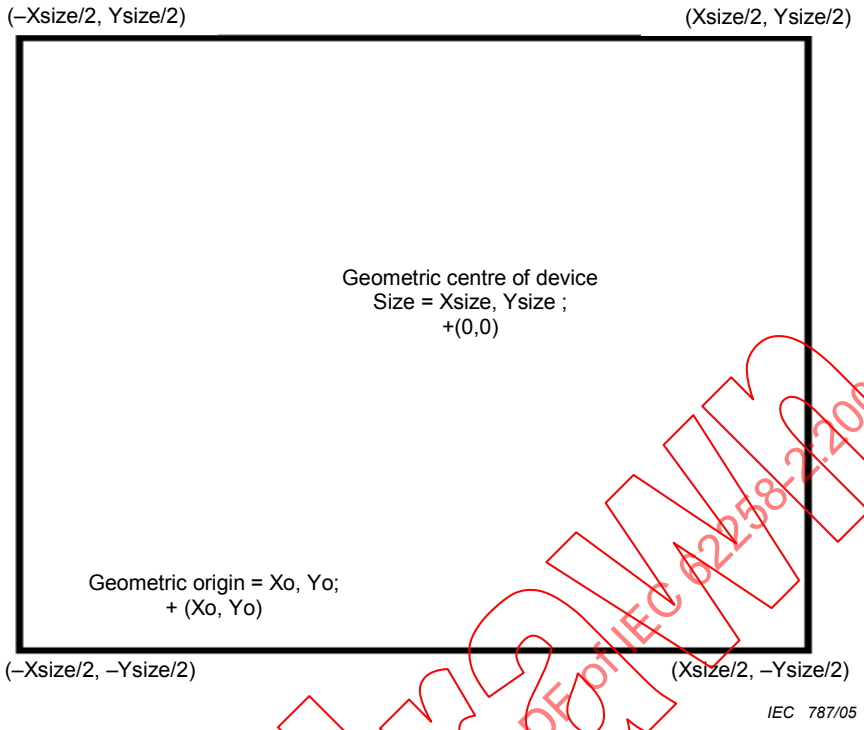

Figure 1 – Relationship between geometric centre and geometric origin

The **GEOMETRIC_ORIGIN** is treated as an offset for all coordinate data, so that the **GEOMETRIC_ORIGIN** values are ***added*** to ***all*** individual coordinate data pairs to give the X- and Y- position relative to the geometric centre of the device.

The primary use of the **GEOMETRIC_ORIGIN** parameter is to permit the centring of the origin on a terminal or other geometric feature, rather than an arbitrary geometric position, as, in practice, **SIZE** may be subject to an asymmetric tolerance so that all related references to this geometric centre could therefore also be subject to a tolerance error.

8.18 SIZE_TOLERANCE Parameter

Parameter Name	**SIZE_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specify the geometric tolerance(s) of the SIZE parameter values.
Parameter Values	Real in **GEOMETRIC_UNITS**, as described in 7.1.3.3
Dependencies	**GEOMETRIC_UNITS**, **SIZE**, **GEOMETRIC_VIEW**
Limitations	Shall be declared only once within a single **DEVICE** block.

Example	`SIZE_TOLERANCE = 0.5;`
	`SIZE_TOLERANCE = -0.2,0.5;`
	`SIZE_TOLERANCE = -0.2,0.5,-0.1,0.4;`

Notes	One, two or four values may be given: Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value for both the X- and Y-axis. Where two tolerance values are given: the first value shall be taken as the minimum for both the X-axis and the Y-axis, the second value shall be taken as the maximum for both X-axis and the Y-axis. Where four tolerance values are given:- the first value shall be taken as the minimum for the X-axis, the second value shall be taken as the maximum for the X-axis, the third value shall be taken as the minimum for the Y-axis, and the fourth value shall be taken as the maximum for the Y-axis.

8.19 THICKNESS_TOLERANCE Parameter

Parameter Name	**THICKNESS_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specifies the tolerance(s) of the THICKNESS parameter that may be expected due to normal process variations.
Parameter Values	Real in **GEOMETRIC_UNITS**, as described in 7.1.3.3
Dependencies	**GEOMETRIC_UNITS**, **THICKNESS**
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`THICKNESS_TOLERANCE = 2.5;` `THICKNESS_TOLERANCE = -1.2,1.5;`
Notes	One or two values may be given: Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value. Where two tolerance values are given: the first value shall be taken as the minimum, and the second value shall be taken as the maximum.

8.20 TERMINAL_COUNT Parameter

Parameter Name	**TERMINAL_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of electrical terminal or points of connection.
Parameter Values	Integer, as described in 7.1.3.4
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`TERMINAL_COUNT = 44;`
Notes	The **TERMINAL_COUNT** value shall be declared before any **TERMINAL** declaration or usage.

8.21 TERMINAL_TYPE_COUNT Parameter

Parameter Name	**TERMINAL_TYPE_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of different connection or bond terminal types or shapes.
Parameter Values	Integer, as described in 7.1.3.4
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`TERMINAL_TYPE_COUNT = 4;`
Notes	The **TERMINAL_TYPE_COUNT** value shall be declared before any **TERMINAL_TYPE** declaration or usage.

8.22 CONNECTION_COUNT Parameter

Parameter Name	**CONNECTION_COUNT**
Parameter Type	Variable, optional
Parameter Function	Specifies the maximum number of connections used by the **TERMINAL** parameter. This parameter actually specifies the highest value for the connection ***conn_N*** numbers used in the **TERMINAL** parameter declaration (refer to 8.24.2).
Parameter Values	Integer, as described in 7.1.3.4
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`CONNECTION_COUNT = 4;`
Notes	The **CONNECTION_COUNT** value should be declared before any **TERMINAL** declaration or usage. It shall be used as a limit check for the connection ***conn_N*** numbers. If the **CONNECTION_COUNT** parameter is not declared, then no check will take place.

8.23 TERMINAL_TYPE Parameter

Parameter Name	**TERMINAL_TYPE**
Parameter Type	Structure
Parameter Function	The **TERMINAL_TYPE** structure assigns a type name and defines the shape and size of an individual terminal type. As a structure, **TERMINAL_TYPE** may be used to define a single terminal type, or a multiple of terminal types.
Dependencies	**GEOMETRIC_UNITS**,**GEOMETRIC_VIEW**, **TERMINAL_TYPE_COUNT**
Notes	The coordinate pairs are relative only to the terminal shape type, and are used as references when placing the shape. Refer to Annex E.

Single **TERMINAL_TYPE** definition syntax:

TERMINAL_TYPE *Terminal_type_name* = *Terminal_shape_type*, *Coordinates*.,,,;
TERMINAL_TYPE *Terminal_type_name* = *Terminal_shape_type*, *Coordinates*.,,,;

Multiple **TERMINAL_TYPE** definition syntax:

TERMINAL_TYPE {
Terminal_type_name = *Terminal_shape_type*, *Coordinates* .,,,;
Terminal_type_name = *Terminal_shape_type*, *Coordinates* .,,,;
Terminal_type_name = *Terminal_shape_type*, *Coordinates* .,,,;
}

where:

8.23.1 Terminal_type_name

Textual reference name for a terminal type, as described in 7.1.3.2, which shall be unique within the **DEVICE** block.

8.23.2 Terminal_shape_type

Determines the terminal shape type (only the first letter need be used):

Table 1 – Terminal shape types

Clause	Char	DET data	Shape
8.23.2.1	R	RECT	Rectangle
8.23.2.2	C	CIRC	Circle
8.23.2.3	E	ELL	Ellipse
8.23.2.4	P	POLY	Polygon

8.23.3 Coordinates

Real coordinate values (refer to Table 2), as described in 7.1.6.

This specifies the relative coordinates for the terminal shape, and in doing so, determines the geometric reference centre for the shape. The geometric reference centre of the shape will be used by the TERMINAL coordinates (8.24.4 and 8.24.5) to place the shape, and all rotation and mirroring will be about this geometric reference centre. Only the polygon shape can have a geometric reference centre other than the natural "centre-of gravity".

Table 2 – Terminal shape coordinates

Clause	Shape	Coordinates	Geometric reference centre
8.23.3.1	Rectangle	X-size, Y-size	The geometric centre (0,0) will occur at X-size/2:Y-size/2
8.23.3.2	Circle	Diameter	The geometric centre (0,0) will occur at diameter/2:diameter/2
8.23.3.3	Ellipse	X-axis, Y-axis	The geometric centre (0,0) will occur at X-axis/2, Y-axis /2
8.23.3.4	Polygon	$X\text{-}co_1$, $Y\text{-}co_1$ ..... $X\text{-}co_N$, $Y\text{-}co_N$	An "N" sided polygon will have N pairs of coordinates, geometrically centred upon (0,0), with the assumption that the polygon shape will be completed with the vector (Xco_N,Yco_N – Xco_1,Yco_1)

Example 1

```
TERMINAL_TYPE ShapeR1 = Rectangle,  200, 250;
```

This example describes a rectangle, uniquely referred to as ShapeR1, with an X-axis dimension of 200 units, and a Y-axis dimension of 250 units. The units are defined by the **GEOMETRIC_UNITS** parameter.

Example 2

```
TERMINAL_TYPE  {
      ShapeR1 = R, 200, 300;
      ShapeC2 = C, 250;
      ShapeE3 = E, 400, 200;
      ShapeP4  =  P,  (150,  200),(-150,  200),(-150,-150),(  150,-
150);
}
```

This example describes four separate terminal types and shapes:

- ShapeR1 is a rectangular terminal having an X-axis dimension of 200 units and a Y-axis dimension of 300 units,
- ShapeC2 is a circular terminal of 250 units in diameter,

- ShapeE3 is an elliptical terminal with the X-axis diameter of 400 units and Y-axis diameter of 200 units, and
- ShapeP4 is a 4 sided polygonal terminal shape, with a geometric reference centre of (0,0).

Note that only the first letter of the *Terminal_shape_type* descriptor is used, and that the *Terminal_type_names* are unique.

8.24 TERMINAL Parameter

Parameter Name	**TERMINAL**
Parameter Type	Structure
Parameter Function	The **TERMINAL** structure defines the name, location, orientation and electrical function of each individual connection terminal. As a structure, **TERMINAL** may be used to define a single terminal, or a multiple of terminals.
Dependencies	**TERMINAL_COUNT**, **CONNECTION_COUNT**, **GEOMETRIC_VIEW**, **TERMINAL_TYPE**, **GEOMETRIC_UNITS**, **GEOMETRIC_ORIGIN**.
Notes	Each terminal type name used shall have been previously declared in a **TERMINAL_TYPE** invocation, and coordinate pair information shall relate to the **GEOMETRIC_ORIGIN** of the device. Refer to Annex E.

Single **TERMINAL** definition syntax:

```
TERMINAL T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
TERMINAL T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
```

Multiple **TERMINAL** definition syntax:

```
TERMINAL {
          T_n = conn_N, Terminal_type_name, X-co., Y-co.,
                orient., Terminal_name, IO_type;
          T_n = conn_N, Terminal_type_name, X-co., Y-co.,
                orient., Terminal_name, IO_type;
          T_n = conn_N, Terminal_type_name, X-co., Y-co.,
                orient., Terminal_name, IO_type;
        }
```

where:

8.24.1 T_n

Unique terminal identifier, where "***n***" is the actual terminal number (Integer, as described in 7.1.3.4), numbering from top left anti-clockwise. Note that the underscore character is optional, it is used for clarity only.

8.24.2 conn_N

Connection number, non-unique integer, as described in 7.1.3.4. This connection number is merely a place reference, and may optionally refer to a package pin; need not be present, or may be zero, 0, or left blank if unknown. The singular advantage of this connection number is in specifying and identifying multiple terminals that need to be connected together. The value of this connection number should be checked to ensure that it does not exceed the value determined by **CONNECTION_COUNT**, when specified.

8.24.3 Terminal_type_name

Textual name, as described in 7.1.3.2, of a referenced **TERMINAL_TYPE** terminal shape (as 8.23.1), which shall have been previously declared, (refer to 7.1.4).

8.24.4 X-co

Numeric real X coordinate value, as described in 7.1.3.3, for location of centre of the terminal shape, in units defined by the **GEOMETRIC_UNITS** parameter. This value is relative to the X coordinate value of the **GEOMETRIC_ORIGIN**.

8.24.5 Y-co

Numeric real Y coordinate value, as described in 7.1.3.3, for location of centre of the terminal shape, in units defined by the **GEOMETRIC_UNITS** parameter. This value is relative to the Y coordinate value of the **GEOMETRIC_ORIGIN**.

8.24.6 Orient

Orientation value, of integer values from 0 to 360, as described in 7.1.3.4. This is the angle of clockwise rotation in degrees about the geometric reference centre of the terminal shape. If the letters "**MX**" are included, then the orientation of the terminal shape shall be mirrored in the X-axis, similarly if the letters "**MY**" are included, then the orientation of the terminal shape shall be mirrored in the Y-axis. Both "**MX**" and "**MY**" may be present simultaneously (refer to Annex D. for a graphical representation), and all mirroring shall be done about the geometric reference centre of the terminal shape. Note that the mirroring operation shall be carried out first, then the terminal shall be rotated by the orientation angle.

8.24.7 Terminal_name

Textual name, non-unique, as described in 7.1.3.2. This terminal name is for user reference and graphic display only, and may be omitted.

8.24.8 IO_type

A single letter indicating the terminal pin function type as given in Table 3, not mandatory. Further characters may be added as required.

Table 3 – Terminal IO types

Clause	Letter	Terminal function
8.24.8.1	**I**	**I**nput, digital
8.24.8.2	**O**	**O**utput, digital
8.24.8.3	**B**	**B**i-directional, digital
8.24.8.4	**G**	**G**round connection
8.24.8.5	**V**	Supply, may be any supply type
8.24.8.6	**A**	**A**nalog pin, input or output
8.24.8.7	**N**	**N**o-connect pin, leave disconnected
8.24.8.8	**U**	**U**ndetermined, user programmable I/O
8.24.8.9	**T**	**T**est pin, connect only under manufacturers advice
8.24.8.10	**X**	Internally connected, do not connect.
8.24.8.11	**H**	Digital functional input pin, to be held at a logic **H**igh
8.24.8.12	**L**	Digital functional input pin, to be held at a logic **L**ow

T_*n* (8.24.1) shall always be unique, whereas the pin connection number (8.24.2), ***conn_N***, and terminal name (8.24.7), ***Terminal_name***, need not be unique.

Example 1

```
TERMINAL T_7 = 9, ShapeR1, (5000, 7000), MX0, VCC1, V;
```

This example declares that terminal **T_7**.....

- is connected to arbitrary connection # 9,
- is located at x = 5000 units, Y = 7000 units,
- is a TERMINAL_TYPE named “ShapeR1”, mirrored on the X-axis, orientated at 0°, and
- is called “VCC1”, and
- is a power supply pin.

Example 2

```
TERMINAL T_8 = 17, ShapeP2, 5000, 7300, 90, qd2i, I;
TERMINAL T_9 = 17, ShapeP2, 5000, 7800, 90, qd2o, O;
```

These example declare that terminals **T_8** & **T_9**.....

- are both connected to arbitrary connection # 17,
- are located at X = 5 000 units, Y = 7 300 units and X = 5000 units, Y = 7800 units respectively,
- are both a **TERMINAL_TYPE** named “ShapeP2”, orientated (rotated) by 90° clockwise,
- are named “qd2i” and “qd2o”
- and constitute a digital I/O bi-directional connection, with “qd2i” being an input (I) and “qd2o” being an output (O).

Example 3

```
TERMINAL T_10 = , ShapeP2, 5000, 7600, 0, , X;
```

This example declares that terminal **T_10**.....

- is unconnected (or with no specific connection reference),
- is located at X = 5000 units, Y = 7600 units,
- is a **TERMINAL_TYPE** named “ShapeP2”, orientated at 0°, and
- has no given name and
- is specified as not to be bonded to.

Example 4.

```
TERMINAL T_44 = , ShapeR2, 25000, 97000, MXMY90, ,;
```

This example declares that terminal **T_44**.....

- is unconnected (or with no specific connection reference),
- is located at (25000:97000),
- is a **TERMINAL_TYPE** named “ShapeR2”, mirrored in both the X- and Y-axis, then orientated at (rotated clockwise by) 90°,
- has no given name, and
- has no specified electrical connection properties.

Example 5

```
TERMINAL {
      T_7 = 9, ShapeR1, (5000, 7000), MX0, VCC1, V;
      T_8 = 17, ShapeP2, (5000, 7300), 90, qd2i, I;
      T_9 = 17, ShapeP2, (5000, 7800), 90, qd2o, O;
      T_10 = , ShapeP2, (5000, 7600), 0, , X;
      T_44 = , ShapeR2, (25000, 97000), MXMY90, ,;
}
```

This multiple terminal definition example declares identical terminal data to that shown in Examples 1 to 4.

8.25 IC_TECHNOLOGY Parameter

Parameter Name	**IC_TECHNOLOGY**
Parameter Type	Variable
Parameter Function	Specifies the fabrication technology of the device
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.

Example	`IC_TECHNOLOGY = “CMOS”;`
	`IC_TECHNOLOGY = “bipolar”;`
	`IC_TECHNOLOGY = “bicmos”;`
	`IC_TECHNOLOGY = “GaAs”;`

8.26 DIE_SEMICONDUCTOR_MATERIAL Parameter

Parameter Name	**DIE_SEMICONDUCTOR_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the semiconductor material which is used for fabrication of the die.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.

Example	`DIE_SEMICONDUCTOR_MATERIAL = “Insulator”;`
	`DIE_SEMICONDUCTOR_MATERIAL = “Silicon”;`
	`DIE_SEMICONDUCTOR_MATERIAL = “Sapphire”;`

Notes	This parameter is only relevant if the die bulk material is different from the die substrate material.

8.27 DIE_SUBSTRATE_MATERIAL Parameter

Parameter Name	**DIE_SUBSTRATE_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the bulk or substrate material on which the die is made where it differs from the DIE_SEMICONDUCTOR_MATERIAL.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.

Example	`DIE_SUBSTRATE_MATERIAL = “Silicon”;`
	`DIE_SUBSTRATE_MATERIAL = “Insulator”;`
	`DIE_SUBSTRATE_MATERIAL = “Sapphire”;`

8.28 DIE_SUBSTRATE_CONNECTION Parameter

Parameter Name	**DIE_SUBSTRATE_CONNECTION**
Parameter Type	Variable
Parameter Function	Specifies the electrical connection for the substrate, if any. By fabrication, the substrate may be already electrically connected, which should also be stated. Where connection may be made, the potential point for connection shall also be stated.
Parameter Values	One or more textual string data, as described in 7.1.3.1 The first parameter shall conform to values given in Table 4
Limitations	Shall be declared only once within a single DEVICE block.

Example
```
DIE_SUBSTRATE_CONNECTION = “CONN”,”Ground”;
DIE_SUBSTRATE_CONNECTION = “N/A”;
DIE_SUBSTRATE_CONNECTION = “ISOL”;
DIE_SUBSTRATE_CONNECTION = “OPT”,”Most Negative”;
DIE_SUBSTRATE_CONNECTION = “OPT”,”Most Positive”;
DIE_SUBSTRATE_CONNECTION = “CONN”,”Digital Vdd”;
DIE_SUBSTRATE_CONNECTION = “OPT”,”Analog ground”;
```

Table 4 – Substrate connection parameters

Clause	Parameter value	Function
8.28.1	**CONN**	**Must** be electrically connected
8.28.2	**ISOL**	**Must** be electrically isolated
8.28.3	**OPT**	May be optionally connected
8.28.4	**N/A**	Not applicable
8.28.5	**N/K**	Not known (unknown)

Note When either CONN or OPT is given as the first parameter value, the second parameter value shall be stated comprehensively, sufficiently to be understood for electrical connectivity. Recommended parameter values are: “Most Positive”, “Most Negative”, “GND”, “AGND”, “DGND”, “VSS”, “VDD”, “VEE”, “VCC”, “AVSS”, “AVDD”, “AVEE”, “AVCC”, “DVSS”, “DVDD”, “DVEE”, “DVCC”, “VBIAS”, or “SPECIAL”.
Alternatively, a specific **TERMINAL** name “**T_n**”, as described in 8.24 may be used as the second parameter value for direct connection to the substrate.

8.29 DIE_PASSIVATION_MATERIAL Parameter

Parameter Name **DIE_PASSIVATION_MATERIAL**
Parameter Type Variable
Parameter Function Specifies the die surface passivation material.
Parameter Values Textual string data, as described in 7.1.3.1
Limitations Shall be declared only once within a single **DEVICE** block.
Example
```
DIE_PASSIVATION_MATERIAL = “Silicon Nitride”;
DIE_PASSIVATION_MATERIAL = “Oxy-Nitride”;
DIE_PASSIVATION_MATERIAL = “Polyimide”;
```

8.30 DIE_TERMINAL_MATERIAL Parameter

Parameter Name **DIE_TERMINAL_MATERIAL**
Parameter Type Variable
Parameter Function Specifies the material used for the pad terminations on the die.
Parameter Values Textual string data, as described in 7.1.3.1
Limitations Shall be declared only once within a single **DEVICE** block.
Example
```
DIE_TERMINAL_MATERIAL = “Al”;
DIE_TERMINAL_MATERIAL = “Cu”;
DIE_TERMINAL_MATERIAL = “Al-Ti-Ni-Au”;
```

8.31 DIE_BACK_DETAIL Parameter

Parameter Name	**DIE_BACK_DETAIL**
Parameter Type	Variable
Parameter Function	Specifies the detail finish to the die backside, relevant for different attachment methods / package styles.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.

Example	`DIE_BACK_DETAIL = “Sawn”;`
	`DIE_BACK_DETAIL = “Metallized”;`
	`DIE_BACK_DETAIL = “Backlapped”;`
	`DIE_BACK_DETAIL = “Polished”;`
	`DIE_BACK_DETAIL = “None”;`

8.32 WAFER_SIZE Parameter

Parameter Name	**WAFER_SIZE**
Parameter Type	Variable
Parameter Function	Specifies the wafer diameter.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`WAFER_SIZE = “6 inch”;`
Notes	As this is a dimensional value, but related to the fabrication and not the die size, the **WAFER_SIZE** parameter may be in units different to those defined by the **GEOMETRIC_UNITS** parameter, hence the data are presented as free-form text.

8.33 MAX_TEMP Parameter

Parameter Name	**MAX_TEMP**
Parameter Type	Variable
Parameter Function	Specifies the maximum temperature to which the device may be exposed during any part of die attach or manufacturing process.
Parameter Values	Real, in °C, as described in 7.1.3.3
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`MAX_TEMP = 280;`

8.34 POWER_RANGE Parameter

Parameter Name	**POWER_RANGE**
Parameter Type	Variable
Parameter Function	Specifies the power likely to be dissipated by the device.
Parameter Values	Real, units in Watts, as described in 7..3.3
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`POWER_RANGE = 0.92;`
Notes	This parameter should be used with caution, as without fully specifying all measurement conditions, use of this value can be misunderstood. The value given should indicate the likely maximum power dissipated under “typical” worst case conditions.

8.35 TEMPERATURE_RANGE Parameter

Parameter Name	**TEMPERATURE_RANGE**
Parameter Type	Variable
Parameter Function	Specifies the operation and specification range of the die.
Parameter Values	Minimum (Real, in ^{0}C), Maximum (Real, in °C), as described in 7.1.3.3
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`TEMPERATURE_RANGE = -40, 90;`
Notes	For use with bare die, this parameter should be used with caution, and is only intended to indicate the operational or specified temperature “grade” of the equivalent packaged part.

8.36 Simulator MODEL FILE Parameter

Parameter Name	**SIMULATOR_*simulator*_MODEL_FILE**
Parameter Type	Variable
Parameter Function	Specifies the name of the model file for use with *simulator*.
Parameter Values	Textual string data, as described in 7.1.3.2
Limitations	Shall be declared only once within a **DEVICE** block per *simulator* type.
Example	`SIMULATOR_SPICE_MODEL_FILE = “BC109.MOD”` `SIMULATOR_SPECTRE_MODEL_FILE = “RTBAA1.S”`
Notes	Only file names, without relative or absolute path names, shall be used. Refer to Annex D.

8.37 Simulator MODEL FILE DATE Parameter

Parameter Name	**SIMULATOR_*simulator*_MODEL_FILE_DATE**
Parameter Type	Variable
Parameter Function	Specifies the creation or validation date of the model file.
Parameter Values	Date, as described in 7.1.3.5
Limitations	Shall be declared only once within a **DEVICE** block per *simulator* type.
Example	`SIMULATOR_SPICE_MODEL_FILE_DATE=”19951021”;` `SIMULATOR_VHDL_MODEL_FILE_DATE=”1993-05-17”;`
Notes	This date should match the date of the actual model file, to indicate that the correct model file is being used, and for use within a library make / build function. Refer to Annex D.

8.38 Simulator NAME Parameter

Parameter Name	**SIMULATOR_*simulator*_NAME**
Parameter Type	Variable
Parameter Function	Specifies the name of either the generic or specific *simulator* to which the model file is appropriate.
Parameter Values	Textual string data, as described in 7.1.3.2
Limitations	Shall be declared only once within a **DEVICE** block per *simulator* type.
Example	`SIMULATOR_SPICE_NAME  = “pSpice”;` `SIMULATOR_VERILOG_NAME  = “Verilog-XL”;`
Notes	Refer to Annex D.

8.39 Simulator VERSION Parameter

Parameter Name	**SIMULATOR_*simulator*_VERSION**
Parameter Type	Variable
Parameter Function	Specifies the version number of the *simulator* as specified by the **SIMULATOR_*simulator*_NAME** parameter, which was used to verify the model data.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a **DEVICE** block per *simulator* type.
Example	`SIMULATOR_SPICE_VERSION  = “4.0.1”;` `SIMULATOR_VERILOG_VERSION  = “1.7B”;`
Notes	Refer to Annex D.

8.40 Simulator COMPLIANCE Parameter

Parameter Name	**SIMULATOR_*simulator*_COMPLIANCE**
Parameter Type	Variable
Parameter Function	Specifies the minimum compliance level of the *simulator* required to both accurately reproduce and correlate with simulation results.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a **DEVICE** block per *simulator* type.
Example	`SIMULATOR_VHDL_COMPLIANCE = "VHDL '93";` `SIMULATOR_SPICE_COMPLIANCE = "2G6";`
Notes	Refer to Annex D.

8.41 DIE_DELIVERY_FORM Parameter

Parameter Name	**DIE_DELIVERY_FORM**
Parameter Type	Variable
Parameter Function	Specifies the form in which the die is delivered.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.

Example	`DIE_DELIVERY_FORM = "Wafer";`
	`DIE_DELIVERY_FORM = "Die";`
	`DIE_DELIVERY_FORM = "Sawn Wafer";`

8.42 PACKING_CODE Parameter

Parameter Name	**PACKING_CODE**
Parameter Type	Variable
Parameter Function	Specifies the form of primary packing used for the supply of devices.
Parameter Values	Textual string data, as described in 7.1.3.1
Example	`PACKING_CODE = "WAFFLE";`

8.43 BUMP_MATERIAL Parameter

Parameter Name	**BUMP_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the metallurgical material of the bump contact.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`BUMP_MATERIAL = "Copper";` `BUMP_MATERIAL = "Molybdenum Telluride";`
Notes	Required only for device types with bump connection structures.

8.44 BUMP_HEIGHT Parameter

Parameter Name	**BUMP_HEIGHT**
Parameter Type	Variable
Parameter Function	Specifies the bump contact height above the passivation surface.
Parameter Values	Numeric real, in **GEOMETRIC_UNITS**, as described in 7.1.3.3
Dependencies	**GEOMETRIC_UNITS**
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`BUMP_HEIGHT = 150.0;`
Notes	Required only for device types with bump connection structures.

8.45 BUMP_HEIGHT_TOLERANCE Parameter

Parameter Name	**BUMP_HEIGHT_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specifies the tolerance of bump contact height above the passivation surface.
Parameter Values	Numeric real, in **GEOMETRIC_UNITS**, as described in 7.1.3.3
Dependencies	**GEOMETRIC_UNITS**, **BUMP_HEIGHT**
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`BUMP_HEIGHT_TOLERANCE = 35.0;` `BUMP_HEIGHT_TOLERANCE = -10.0,25.0;`
Notes	one or two values may be given:- Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value. Where two tolerance values are given:- the first value shall be taken as the minimum, and the second value shall be taken as the maximum. Required only for device types with bump connection structures.

8.46 MPD_PACKAGE_MATERIAL Parameter

Parameter Name	**MPD_PACKAGE_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the package body material used as final encapsulation of the MPD.
Parameter Values	Textual string data as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.

Example	`MPD_PACKAGE_MATERIAL = “Plastic”;`
	`MPD_PACKAGE_MATERIAL = “Epoxy”;`
	`MPD_PACKAGE_MATERIAL = “Ceramic”;`
	`MPD_PACKAGE_MATERIAL = “Metal Can”;`

Notes	Generally required for assembly purposes.

8.47 MPD_PACKAGE_STYLE Parameter

Parameter Name	**MPD_PACKAGE_STYLE**
Parameter Type	Variable
Parameter Function	Specifies the code for the package style as defined in a standard or as given by the manufacturer.
Parameter Values	Textual string data as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.

Example	`MPD_PACKAGE_STYLE = “SOT-23”;`
	`MPD_PACKAGE_STYLE = “MO”;`
	`MPD_PACKAGE_STYLE = “TSOP-48”;`

8.48 MPD_DELIVERY_FORM Parameter

Parameter Name	**MPD_DELIVERY_FORM**
Parameter Type	Variable(s)
Parameter Function	Specifies the delivery media that is used for this MPD device, more than one media may be available.
Parameter Values	Textual string data, as described in 7.1.3.1
Example	`MPD_DELIVERY_FORM = “Bandoleer”, “Waffle”;`

8.49 MPD_CONNECTION_TYPE Parameter

Parameter Name	**MPD_CONNECTION_TYPE**
Parameter Type	Variable
Parameter Function	Specifies the connection method to the MPD, such as lead, bump etc.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`MPD_CONNECTION_TYPE = “Bump”;`

8.50 MPD_CONNECTION_MATERIAL Parameter

Parameter Name	**MPD_CONNECTION_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the connection material of the MPD.
Parameter Values	Textual string data, as described in 7.1.3.1
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`MPD_CONNECTION_MATERIAL = “Gold”;` `MPD_CONNECTION_MATERIAL = “Pb-Sn”;`

8.51 FIDUCIAL_TYPE Parameter

Parameter Name	**FIDUCIAL_TYPE**
Parameter Type	Structure
Parameter Function	The **FIDUCIAL_TYPE** structure assigns a fiducial type name and defines the associated graphic file and size of an individual fiducial type. The structure may be used to define a single fiducial type, or a multiple of fiducial types.
Dependencies	**GEOMETRIC_UNITS**, **GEOMETRIC_VIEW**
Notes	A fiducial only exists as a graphic shape within a rectangle, the graphic data for the fiducial is held within an external graphic file. The coordinate pairs for this rectangle are relative only to the fiducial shape type, and are used as references when placing the shape. Refer to Annex E.

Single FIDUCIAL_TYPE definition syntax:

FIDUCIAL_TYPE *Fiducial_type_name* = *Fiducial_file_name*, *X-size, Y-size;*
FIDUCIAL_TYPE *Fiducial_type_name* = *Fiducial_file_name*, *X-size, Y-size;*

Multiple FIDUCIAL_TYPE definition syntax:

FIDUCIAL_TYPE {
Fiducial_type_name = *Fiducial_file_name*, *X-size, Y-size;*
Fiducial_type_name = *Fiducial_file_name*, *X-size, Y-size;*
Fiducial_type_name = *Fiducial_file_name*, *X-size, Y-size;*
}

where:

8.51.1 Fiducial_type_name

Textual reference name for a fiducial type, as described in 7.1.3.2, which shall be unique within the **DEVICE** block.

8.51.2 Fiducial_file_name

This is the name of the file that holds the fiducial as graphic data. The graphic data type shall be indicated by the file extension code, such as “BMP”, “GIF”, “DXF” etc. The data type is not specified with this standard, and it is up to the CAD/CAM software as to what can and cannot be displayed. The graphic shall be treated as being contained within a rectangle of X-size, Y-size dimensions.

8.51.3 X-size, Y-size

These real numeric parameters comprise a coordinate pair, they determine the rectangle size of the fiducial graphic and are identical to those described in 8.23.3.1. The geometric centre of the fiducial graphic shall be determined as being (X-size/2, Y-size/2).

Example 1.

```
FIDUCIAL_TYPE Fid1 = “tile.bmp”, 200, 250;
```

This example describes a fiducial, found as an external file “`tile.bmp`”, uniquely referred to as `Fid1`, with an X-size of 200 units, and a Y-size of 250 units. The units are defined by the **GEOMETRIC_UNITS** parameter.

Example 2.

```
FIDUCIAL_TYPE  {
      fidu1 = “graphic1.bmp”, 200, 300;
      fidu2 = “graphic2.dxf”, 500, 500;
      fidu3 = “graphic3.gif”, 100, 100;
}
```

This multiple definition example describes three separate fiducial types and shapes:

- `fidu1` is contained in file “`graphic1.bmp`”, of size 200 by 300 units.
- `fidu2` is contained in file “`graphic2.dxf`”, of size 500 by 500 units.
- `fidu3` is contained in file “`graphic3.gif`”, of size 100 by 100 units.

8.52 FIDUCIAL Parameter

Parameter Name	**FIDUCIAL**
Parameter Type	Structure
Parameter Function	The **FIDUCIAL** structure defines the location and orientation of each individual graphic fiducial. As a structure, **FIDUCIAL** may be used to define a single fiducial, or a multiple of fiducials.
Dependencies	**GEOMETRIC_VIEW**, **FIDUCIAL_TYPE**, **GEOMETRIC_UNITS**, **GEOMETRIC_ORIGIN**.
Notes	Each fiducial type name used shall have been previously declared in a **FIDUCIAL_TYPE** invocation, and coordinate pair information shall relate to the **GEOMETRIC_ORIGIN** of the device. Refer to Annex E.

Single **FIDUCIAL** definition syntax:

FIDUCIAL F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
FIDUCIAL F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*

Multiple FIDUCIAL definition syntax:

FIDUCIAL {
F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
}

where:

8.52.1 F_*n*

Unique fiducial identifier, where “***n***” is the actual fiducial number (Integer, as described in 7.1.3.4), numbering from top left anti-clockwise. Note that the underscore character is optional, it is used for clarity only.

8.52.2 Fiducial_type_name

Textual name, as described in 7.1.3.2 of a referenced **FIDUCIAL_TYPE** shape (as 8.51.1), which shall have been previously declared, (refer to 7.1.4).

8.52.3 X-co

Numeric real X coordinate value, as described in 7.1.3.3, for location of the centre of the fiducial shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the X coordinate value of the GEOMETRIC_ORIGIN, and this parameter is identical in operation to that described in 8.24.4.

8.52.4 Y-co

Numeric real Y coordinate value, as described in 7.1.3.3, for location of the centre of the fiducial shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the Y coordinate value of the GEOMETRIC_ORIGIN, and this parameter is identical in operation to that described in 8.24.5.

8.52.5 orientation

Orientation value, of integer values from 0 to 360, as described in 7.1.3.4. This is the angle of clockwise rotation in degrees about the geometric reference centre of the fiducial shape. If the letters "**MX**" are included, then the orientation of the fiducial shape shall be mirrored in the X-axis, similarly if the letters "**MY**" are included, then the orientation of the fiducial shape shall be mirrored in the Y-axis. Both "**MX**" and "**MY**" may be present simultaneously (refer to Annex D for a graphical representation), and all mirroring shall be done about the geometric reference centre of the fiducial shape. Note that the mirroring operation shall be carried out first, then the fiducial shall be rotated by the orientation angle. This parameter is identical in operation to that described in 8.24.6

Example 1

```
FIDUCIAL F_7 = fidu1, 5000, 7000, MX0;
```

This example declares that fiducial **F_7**.....

- is located at X = 5000 units, Y = 7000 units,
- is a FIDUCIAL_TYPE named "fidu1", mirrored on the X-axis and orientated at 0°.

Example 2

```
FIDUCIAL F_18 = fiduX1, 1000, 2200, 90;
```

This example declares that fiducial **F_18** .....

- is located at X = 1000 units, Y = 2200 units
- is a **FIDUCIAL_TYPE** named "fiduX1", orientated (rotated) by 90° clockwise.

Example 3

```
FIDUCIAL {
  F_7  = fidu1,  5000, 7000, MX0;
  F_18 = fiduX1, 1000, 2200, 90;
}
```

This multiple fiducial definition example declares identical fiducial data to that shown in Examples 1 and 2.

8.53 WAFER_DIE_STEP_SIZE Parameter

Parameter Name	**WAFER_DIE_STEP_SIZE**
Parameter Type	Variable
Parameter Function	Determines the X- and Y- step size dimensions, in **GEOMETRIC_UNITS,** for the die on wafer, as required by mechanical handling equipment.
Parameter Values	Real X-dimension, Real Y-dimension, (as described in 7.1.3.3)
Dependencies	**GEOMETRIC_UNITS**, **GEOMETRIC_VIEW**
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`WAFER_DIE_STEP_SIZE = 280, 530;`
Notes	Refer to Annex H.

8.54 WAFER_GROSS_DIE_COUNT Parameter

Parameter Name	**WAFER_GROSS_DIE_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of whole and viable gross die (of the die type in question) available on the wafer.
Parameter Values	Integer, as described in 7.1.3.4
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`WAFER_GROSS_DIE_COUNT = 2027;`
Notes	This parameter value is only intended to give an indication of the number of relevant and viable die per wafer, and shall have no other implication. Refer to Annex H.

8.55 WAFER_INDEX Parameter

Parameter Name	**WAFER_INDEX**
Parameter Type	Variable
Parameter Function	Specifies the type of index feature on a wafer which acts as a reference feature and its orientation with respect to the X-axis for the die. The first parameter determines the index feature type, and the second parameter specifies the approximate angular relationship between the assumed X-axis for the die and the index feature on the wafer. The angle shall be given, in degrees counted clockwise, of the index feature, taking the X-axis of the die as the reference.
Parameter Values	Textual string data, as described in 7.1.3.1, with value “Flat” or “Notch”, followed by a single integer value, in units of degrees, from 0 to 359, as described in 7.1.3.4.
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`WAFER_INDEX = “Flat”,90;`
Notes	This parameter is only intended as a guide, and so integer approximations shall be acceptable. Negative angle values are not permitted. Where more than one flat exists on the wafer, the orientation is with respect to the major, or prime, flat. This may also indicate the crystal [110] direction.

Refer to Annex H.

8.56 WAFER_RETICULE_STEP_SIZE Parameter

Parameter Name	**WAFER_RETICULE_STEP_SIZE**
Parameter Type	Variable
Parameter Function	Specifies X- and Y- step and repeat dimensions for a single reticule.
Parameter Values	Real X-dimension, Real Y-dimension, (as described in 7.1.3.3)
Dependencies	**GEOMETRIC_UNITS**, **GEOMETRIC_VIEW**
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`WAFER_RETICULE_STEP_SIZE = 2500, 8000;`
Notes	This parameter is relevant mainly for MPW (Multi Project Wafers), or instances where the WAFER_DIE_STEP_SIZE becomes invalid due to a non-integer relationship between reticule size and die size. Refer to Annex H.

8.57 WAFER_RETICULE_GROSS_DIE_COUNT Parameter

Parameter Name	**WAFER_RETICULE_GROSS_DIE_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of whole and viable gross die (of the die type in question) emplaced within a single reticule.
Parameter Values	Integer, as described in 7.1.3.4
Limitations	Shall be declared only once within a single **DEVICE** block.
Example	`WAFER_RETICULE_GROSS_DIE_COUNT = 40;`
Notes	This parameter is relevant mainly for MPW (Multi Project Wafers), or instances where there is more than one die type in the reticule. This parameter value is only intended to give an indication of the number of relevant and viable die per reticule, and shall have no other implication. Refer to Annex H.

9 DDX EXPRESS model schema

This clause contains the full EXPRESS listing of the DDX schema, annotated with comments and explanatory text. The order of text in this clause is determined primarily by the order imposed by the EXPRESS language, secondarily by importance.

```
*)
SCHEMA ddx;
(*
```

9.1 Type definitions

This subclause contains definitions for the types used within this EXPRESS model.

```
*)
TYPE variable = STRING;
END_TYPE;

TYPE date_type = STRING;
END_TYPE;

TYPE geometric_unit
= ENUMERATION OF (micron, metre, millimetre, inch, mil);
END_TYPE;


TYPE geometric_view_value
= ENUMERATION OF (top,bottom);
END_TYPE;

(*
** A distance in the corresponding "geometric_unit".
*)
TYPE geometric_value = REAL;
END_TYPE;

TYPE integer_value = INTEGER;
WHERE
  non_negative: SELF >= 0;
END_TYPE;
(*
```

Formal propositions:

non_negative: The integer is non-negative..

```
*)
TYPE angle_value
= INTEGER;
WHERE
  valid_value: {0 <= SELF <= 359};
END_TYPE;
(*
```

Formal propositions:

valid_value: The angle can take values from 0° to 359°

```
*)
TYPE celsius_value
= REAL;
END_TYPE;

TYPE watt_value
= REAL;
END_TYPE;

TYPE device_name_parameter
= variable;
END_TYPE;

TYPE packing_code_parameter
= variable;
END_TYPE;

TYPE wafer_size_parameter
= variable;
END_TYPE;

TYPE wafer_die_step_size_parameter
= size_value;
END_TYPE;

TYPE wafer_gross_die_count_parameter
= integer_value;
END_TYPE;

TYPE wafer_index_type_parameter
= ENUMERATION OF (Flat, Notch);
END_TYPE;

TYPE wafer_index_orientation_parameter
= angle_value;
END_TYPE;

TYPE wafer_reticule_step_size_parameter
= size_value;
END_TYPE;

TYPE wafer_reticule_gross_die_count_parameter
= integer_value;
END_TYPE;
```

```
TYPE ic_technology_parameter
= variable;
END_TYPE;

TYPE data_source_parameter
= variable;
END_TYPE;

TYPE block_version_parameter
= variable;
END_TYPE;

TYPE function_parameter
= variable;
END_TYPE;

TYPE manufacturer_parameter
= variable;
END_TYPE;

TYPE die_passivation_material_parameter
= variable;
END_TYPE;

TYPE die_terminal_material_parameter
= variable;
END_TYPE;

TYPE die_packaged_part_name_parameter
= variable;
END_TYPE;

TYPE die_name_parameter
= variable;
END_TYPE;

TYPE die_semiconductor_material_parameter
= variable;
END_TYPE;

TYPE die_back_detail_parameter
= variable;
END_TYPE;

TYPE die_substrate_material_parameter
= variable;
END_TYPE;

TYPE die_mask_revision_parameter
= variable;
END_TYPE;

TYPE bump_material_parameter
= variable;
END_TYPE;

TYPE mpd_connection_material_parameter
= variable;
END_TYPE;
```

```
TYPE mpd_connection_type_parameter
= variable;
END_TYPE;

TYPE mpd_delivery_form_parameter
= variable;
END_TYPE;

TYPE mpd_package_style_parameter
= variable;
END_TYPE;

TYPE mpd_package_material_parameter
= variable;
END_TYPE;

TYPE simulator_compliance_parameter
= variable;
END_TYPE;

TYPE simulator_version_parameter
= variable;
END_TYPE;

TYPE simulator_name_parameter
= variable;
END_TYPE;

TYPE simulator_model_file_date_parameter
= variable;
END_TYPE;

TYPE simulator_model_file_parameter
= variable;
END_TYPE;

TYPE die_substrate_connection_parameter
= ENUMERATION OF (CONN, ISOL, OPT, NA, NK);
END_TYPE;

TYPE ellipse_parameter
= ENUMERATION OF (E);
END_TYPE;

TYPE io_type
    = ENUMERATION OF (I, O, B, G, V, A, N, U, T, X, H, L);
END_TYPE;
(*
```

9.2 File structure

This subclause contains the two structures defined for the DDX language, DEVICE and DEVICE_BLOCK.

EXPRESS specification:

```
*)
ENTITY ddx_file;
  devices: SET [1:?] OF device;
  device_blocks: SET [1:?] OF device_block;
END_ENTITY;
(*
```

Attribute definitions:

devices: a set of device names.

device_blocks: a set of device blocks containing the die data.

9.3 Device names

This subclause contains the name of a device for inclusion in a set of devices.

EXPRESS specification:

```
*)
ENTITY device;
  device_name: device_name_parameter;
INVERSE
  containing_file: ddx_file FOR devices;
END_ENTITY;
(*
```

Attribute definitions:

device_name: a name by which a device can be identified.

9.4 Device block

This subclause contains the definition of the contents of a device block.

EXPRESS specification:

```
*)
ENTITY device_block
  ABSTRACT SUPERTYPE OF (ONEOF (bare_or_bumped_die,
    lead_frame_die,
  minimally_packaged_device));
  described_device: device; -- reference
  block_creation_date: date;
  block_version: block_version_parameter;
  manufacturer: manufacturer_parameter;
  function: function_parameter;
  data_source: data_source_parameter;
  geometric_units: geometric_unit;
  geometric_view: geometric_view_value;
  size: die_size_and_shape
  thickness: geometric_value;
  geometric_origin: position;
```

```
  minimum_x_tolerance: OPTIONAL geometric_value;
  minimum_y_tolerance: OPTIONAL geometric_value;
  maximum_x_tolerance: OPTIONAL geometric_value;
  maximum_y_tolerance: OPTIONAL geometric_value;
  minimum_thickness_tolerance: OPTIONAL geometric_value;
  maximum_thickness_tolerance: OPTIONAL geometric_value;
  terminal_types: SET [1: ?] OF terminal_type;
  terminals: SET [1: ?] OF terminal;
  ic_technology: OPTIONAL ic_technology_parameter;
  maximum_temperature: OPTIONAL celsius_value;
  power_range: OPTIONAL watt_value;
  minimum_operation_temperature: OPTIONAL celsius_value;
  maximum_operation_temperature: OPTIONAL celsius_value;
  simulation_data: SET [0: ?] OF simulator_data;
  wafer_size: OPTIONAL wafer_size_parameter;
  wafer_die_step_size: OPTIONAL wafer_die_step_size_parameter;
  wafer_gross_die_count: OPTIONAL wafer_gross_die_count_parameter;
  wafer_index: OPTIONAL wafer_index_parameters;
  wafer_reticule_step_size: OPTIONAL
wafer_reticule_step_size_parameter;
  wafer_reticule_gross_die_count:
           OPTIONAL wafer_reticule_gross_die_count_parameter;
  packing_code: OPTIONAL packing_code_parameter;
  fiducial_types: SET [0: ?] OF fiducial_type;
  fiducials: SET [0: ?] OF fiducial;
  connection_count: integer_value;
DERIVE
  terminal_type_count: INTEGER:= SIZEOF (terminal_types);
  terminal_count: INTEGER:= SIZEOF (terminals);
INVERSE
  containing_file: ddx_file FOR device_blocks;
WHERE
  valid_device: described_device IN containing_file.devices;
  terminal_numbers: terminal_type_count <= terminal_count;
END_ENTITY;
(*
```

9.5 Die size

This subclause contains those properties for the size of a rectangular or elliptical die.

EXPRESS specification:

```
*)
ENTITY die_size_and_shape
  die_size: size_value;
  shape: OPTIONAL ellipse_parameter;
END_ENTITY;
```

9.6 Bare or bumped die type

This subclause contains those properties which are common to both bumped and non-bumped bare die.

EXPRESS specification:

```
*)
ENTITY bare_or_bumped_die
  SUBTYPE OF (device_block);
  die_name: die_name_parameter;
  die_mask_revision: OPTIONAL die_mask_revision_parameter;
  die_packaged_part_name: OPTIONAL die_packaged_part_name_parameter;
  die_substrate_material: OPTIONAL die_substrate_material_parameter;
  die_semiconductor_material: OPTIONAL
    die_semiconductor_material_parameter;
  substrate_connection: die_substrate_connection;
  die_back_detail: OPTIONAL die_back_detail_parameter;
  die_delivery_forms: SET[0, ?] OF die_delivery_form;
  die_passivation_material: OPTIONAL
die_passivation_material_parameter;
  die_terminal_material: OPTIONAL die_terminal_material_parameter;
UNIQUE
  u1: described_device;
END_ENTITY;
(*
```

9.7 Bare die type

This subclause contains those properties which are specific to non-bumped bare die.

EXPRESS specification:

```
*)
ENTITY bare_die
  SUBTYPE OF (bare_or_bumped_die)
UNIQUE
  u1: described_device;
END_ENTITY;
(*
```

9.8 Bumped bare die type

This subclause contains those properties which are specific to bumped bare die.

EXPRESS specification:

```
*)
ENTITY bumped_die
  SUBTYPE OF (bare_or_bumped_die);
  bump_material: bump_material_parameter;
  bump_height: geometric_value;
  minimum_bump_height_tolerance: OPTIONAL geometric_value;
  maximum_bump_height_tolerance: OPTIONAL geometric_value;
UNIQUE
  u1: described_device;
END_ENTITY;
(*
```

9.9 Lead-frame die type

This subclause contains those properties which are specific to bare die with attached lead frames.

EXPRESS specification:

```
*)
ENTITY lead_frame_die
  SUBTYPE OF (device_block);
UNIQUE
  u1: described_device;
END_ENTITY;
(*
```

9.10 Minimally packaged device

This subclause contains those properties which are specific to minimally packaged die devices.

EXPRESS specification:

```
*)
ENTITY minimally_packaged_device
  SUBTYPE OF (device_block);
  package_material: OPTIONAL mpd_package_material_parameter;
  package_style: OPTIONAL mpd_package_style_parameter;
  delivery_forms: SET [0: ?] OF mpd_delivery_form_parameter;
  connection_type: mpd_connection_type_parameter;
  connection_material: OPTIONAL mpd_connection_material_parameter;
UNIQUE
  u1: described_device;
END_ENTITY;
(*
```

9.11 Die delivery forms

This subclause contains the property defining the form in which die are delivered.

EXPRESS specification:

```
*)
ENTITY die_delivery_form
  delivery_form: value;
END_ENTITY;
(*
```

9.12 Terminal types

This subclause contains the type and name of a connecting terminal on a die.

EXPRESS specification:

```
*)
ENTITY terminal_type
  ABSTRACT SUPERTYPE OF (ONEOF (rectangle_terminal_type,
    circle_terminal_type, ellipse_terminal_type,
polygon_terminal_type));
  terminal_type_name: variable;
INVERSE
  containing_device_block: device_block FOR terminal_types;
UNIQUE
  name_in_block: terminal_type_name, containing_device_block;
END_ENTITY;
(*
```

9.13 Rectangular terminal

This subclause contains those properties for a rectangular terminal.

EXPRESS specification:

```
*)
ENTITY rectangle_terminal_type
  SUBTYPE OF (terminal_type);
  rectangle_size: size_value;
END_ENTITY;
(*
```

9.14 Circular terminal

This subclause contains those properties for a circular terminal.

EXPRESS specification:

```
*)
ENTITY circle_terminal_type
  SUBTYPE OF (terminal_type);
  diameter: geometric_value;
END_ENTITY;
(*
```

9.15 Elliptic terminal

This subclause contains those properties for an elliptic terminal.

EXPRESS specification:

```
*)
ENTITY ellipse_terminal_type
  SUBTYPE OF (terminal_type);
  axes: size_value;
END_ENTITY;
(*
```

9.16 Polygonal terminal

This subclause contains those properties for a polygonal terminal.

EXPRESS specification:

```
*)
ENTITY polygon_terminal_type
  SUBTYPE OF (terminal_type);
  coordinates: LIST [3:?] OF position;
WHERE
  implicitly_closed:
    coordinates[1] <> coordinates [SIZEOF (coordinates)];
END_ENTITY;
(*
```

Attribute definitions:

coordinates: the list of coordinate pairs which define the vertices of the polygon. There shall be at least three pairs.

Formal propositions:

implicitly_closed: The polygon is closed with a vector from the last vertex to the first vertex which are distinct points.

9.17 Terminals

This subclause contains those properties which define a terminal.

EXPRESS specification:

```
*)
ENTITY terminal;
  terminal_number: variable;
  corresponding_connection: OPTIONAL variable;
  corresponding_terminal_type: terminal_type; -- reference
  location: position;
  orient: orientation;
  terminal_name: OPTIONAL variable;
  pin_function: OPTIONAL io_type;
INVERSE
  containing_device_block: device_block FOR terminals;
UNIQUE
  number_in_block: terminal_number, containing_device_block;
WHERE
  valid_connection:
    corresponding_connection IN containing_device_block.connections;
  valid_terminal_type:
    corresponding_terminal_type IN
      containing_device_block.terminal_types;
END_ENTITY;
(*
```

9.18 Simulation data

This subclause contains those properties for simulation data and the associated simulator.

EXPRESS specification:

```
*)
ENTITY simulator_data;
  simulator_type: variable;
  model_file: simulator_model_file_parameter;
  file_date: OPTIONAL simulator_model_file_date_parameter;
  simulator_name: simulator_name_parameter;
  simulator_version: simulator_version_parameter;
  minimum_compliance_level: OPTIONAL simulator_compliance_parameter;
INVERSE
  containing_device_block: device_block FOR simulation_data;
UNIQUE
  data_for_simulator:
  simulator_type, containing_device_block;
END_ENTITY;
(*
```

9.19 Fiducial type

This subclause contains those properties defining the appearance of a fiducial on a die.

EXPRESS specification:

```
*)
ENTITY fiducial_type;
  fiducial_type_name: variable;
  fiducial_file_name: variable;
  fiducial_size: size_value;
INVERSE
  containing_device_block: device_block FOR fiducial_types;
UNIQUE
  name_in_block: fiducial_type_name, containing_device_block;
END_ENTITY;
(*
```

9.20 Fiducial

This subclause contains those properties defining the position and orientation of a fiducial on a die.

EXPRESS specification:

```
*)
ENTITY fiducial;
  fiducial_number: integer_value;
  corresponding_fiducial_type: fiducial_type; -- reference
  location: position;
  orient: orientation;
INVERSE
  containing_device_block: device_block FOR fiducials;
UNIQUE
  number_in_block: fiducial_number, containing_device_block;
```

```
WHERE
  valid_fiducial_type:
    corresponding_fiducial_type IN
      containing_device_block.fiducial_types;

END_ENTITY;
(*
```

9.21 Die and feature size

This sub-clause contains those properties defining length and width of the die or of a feature on the die.

EXPRESS specification:

```
*)
ENTITY size_value;
  x_size: geometric_value;
  y_size: geometric_value;
INVERSE
  containing_device_block: device_block FOR size;
END_ENTITY;
(*
```

9.22 Position

This subclause contains those properties defining the position coordinates of a terminal or a fiducial on a die.

EXPRESS specification:

```
*)
ENTITY position;
  x: geometric_value;
  y: geometric_value;
INVERSE
  containing_device_block: device_block FOR geometric_origin;
END_ENTITY;
(*
```

9.23 Orientation

This subclause contains those properties defining the orientation of a terminal or a fiducial on a die.

EXPRESS specification:

```
*)
ENTITY orientation;
  rotation: angle_value;
  mirror_x: BOOLEAN;
  mirror_y: BOOLEAN;
END_ENTITY;
(*
```

9.24 Date

This subclause contains the definition of dates.

EXPRESS specification:

```
*)
ENTITY date;
  ISO_date: date_type;
END_ENTITY;
(*
```

9.25 Substrate connection

This subclause contains those properties defining the connection requirement for the die substrate.

EXPRESS specification:

```
*)
ENTITY die_substrate_connection;
  connection_reqirement: die_substrate_connection_parameter;
  connection_point: OPTIONAL STRING;
END_ENTITY;
```

9.26 Wafer index

This subclause contains those properties defining the type and position of an index on a wafer.

EXPRESS specification:

```
*)
ENTITY wafer_index_parameters;
  wafer_index_type: wafer_index_type_parameter;
  wafer_index_orientation: wafer_index_orientation_parameter;
END_ENTITY;

END_SCHEMA;
(*
```

Annex A
(informative)

Example of a DDX DEVICE block

```
DEVICE 7995 bare_die {
#
# Initial header data, with block and device history.
#
BLOCK_CREATION_DATE = “2000-12-25”;
BLOCK_VERSION = 1.0;
MANUFACTURER = “Fuzziwuzz Logic Ltd.”;
FUNCTION = “Special gate”;
DATA_SOURCE = “GOOD-DIE database”;
DATA_VERSION = “Initial Issue A”;
VERSION = “1.2.1”;

#
# Declaration of geometric view, units, size etc.,
#
GEOMETRIC_UNITS = millimetre;
GEOMETRIC_VIEW = “top”;
SIZE = 1.312, 1.050;
SIZE_TOLERANCE = 0.00, 0.0005, 0.00 0.0005;
THICKNESS = 0.360;
THICKNESS_TOLERANCE = 0.00, 0.0007;
GEOMETRIC_ORIGIN = 0,0;

#
# Additional details of Die type and usage.
#
DIE_NAME = “XXZ322”;
DIE_MASK_REVISION = “Mask 1.0”;
MAX_TEMP = 280;
POWER_RANGE = 0.500;
DIE_SUBSTRATE_MATERIAL = “Silicon”;
DIE_TERMINAL_MATERIAL = “Al”;
IC_TECHNOLOGY = “bipolar”;
DIE_SUBSTRATE_CONNECTION = “Ground”;

#
# Delivery details.
#
DIE_BACK_DETAIL  = “Back-Lapped”;
DIE_DELIVERY_FORM = “Die, Wafer”;
WAFER_SIZE = “4 inch”;

#
# Definition of the number of bond pad types, bond pads
# and connections.
#
TERMINAL_TYPE_COUNT    = 5;
TERMINAL_COUNT         = 8;
CONNECTION_COUNT       = 14;
```

```
#
# Definition of the bond pad shapes and dimensions.
#
TERMINAL_TYPE  {
      PADR1 = Rectangle, 0.144, 0.104;
      PADR2 = Rectangle, 0.264, 0.104;
      PADR3 = Rectangle, 0.084, 0.084;
      PADC1 = Circle, 0.100;
      PADP1 = Polygon, (-0.0175,-0.042),(-0.042,-0.0175),
      (-0.042, 0.0175),(-0.0175, 0.042),
      ( 0.0175, 0.042),( 0.042, 0.0175),
      ( 0.042,-0.0175),( 0.0175,-0.042);
      }

#
# Bond pad placement, naming, orientation and connectivity
# details.
#
TERMINAL {
      T1 = 1 ,PADC1,-0.550, 0.416,  0,VCCA   ,P;
      T2 = 3 ,PADP1,-0.502, 0.190,  0,INPUTA ,I;
      T3 = 4 ,PADP1,-0.502,-0.192,  0,INPUTB ,I;
      T4 = 7 ,PADC1,-0.399,-0.442,  0,GNDA   ,G;
      T5 = 8 ,PADR2, 0.498,-0.442,  0,GNDB   ,G;
      T6 = 11,PADR3, 0.511,-0.171,  0,OUTPUTA,O;
      T7 = 12,PADR3, 0.511, 0.171,  0,OUTPUTB,O;
      T8 = 14,PADR1, 0.558, 0.416,  0,VCCB   ,P;
       }

#
# Details of a supplied pSpice simulation model.
#
SIMULATOR_SPICE_MODEL_FILE = “SP7995.MOD”;
SIMULATOR_SPICE_MODEL_FILE_DATE = “1997-09-17”;
SIMULATOR_SPICE_NAME = “pSpice”;
SIMULATOR_SPICE_VERSION = “4.0.1”;
SIMULATOR_SPICE_COMPLIANCE = “2G6”;

#
# Details of a supplied Spectre simulation model.
#
SIMULATOR_SPECTRE_MODEL_FILE = “SP7995.S”;
SIMULATOR_SPECTRE_MODEL_FILE_DATE = “1998-11-05”;
SIMULATOR_SPECTRE_NAME = “Spectre”;
SIMULATOR_SPECTRE_VERSION = “4.2.1, 1992”;
SIMULATOR_SPECTRE_COMPLIANCE = “2G6, Level-3”;

#
# Details of reference fiducial, supplied as a JIF graphic file
#
FIDUCIAL_TYPE fiduc1 = “7995FID1.JIF”, 0.072, 0.055;
FIDUCIAL F1 = fiduc1, -0.612, 0.470, 0;

# End of block
}
```

Annex B
(informative)

Example of DDX data in STEP Physical FILE (SPF) format

The following gives the die data in STEP Physical File format in accordance with the EXPRESS model in Clause 9 for the example data set used in Annex A .

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('DDX example'),'1.2.1');
FILE_NAME('SPF example.SPF','2002-01-30T10:40:00+00:00',
  ('D E Radley'),( ), '','','D E Radley');
FILE_SCHEMA(('DDX'));
ENDSEC;

DATA;

#1=DDX_FILE((#2), (#3));
#2=DEVICE_NAME('7995');
#3=BARE_DIE(#2, '2002-01-30', '1.0', 'Fuzziwuz Logic Ltd',
  'Special gate', 'GOOD-DIE database', 'mm', 'top', #40, 0.360, #41,
  $, $, $, $, $, $, (#10, #11, #12, #13, #14), (#20, #21, #22,
  #23, #24, #25, #26, #27), 'bipolar', 280, 0.500, $, $, (#60, #61),
  '4 inch', $, (#70), (#71), $, $, $, $, $, $, 14, *, *, 'XXZ322',
  'Mask 1.0', $, 'Silicon', $, #30, 'Back-lapped', (#4, #5), $,
  'Al');
#4=DIE_DELIVERY_FORM('Die');
#5=DIE_DELIVERY_FORM('Wafer');

#10=RECTANGLE_TERMINAL_TYPE('PADR1', 0.144, 0.104);
#11=RECTANGLE_TERMINAL_TYPE('PADR2', 0.264, 0.104);
#12=RECTANGLE_TERMINAL_TYPE('PADR3', 0.084, 0.104);
#13=CIRCLE_TERMINAL_TYPE('PADC1', 0.100);
#14=POLYGON_TERMINAL_TYPE('PADP1', (#42, #43, #44, #45, #46,
  #47, #48, #49));

#20=TERMINAL( T1,  1, #13, #51, #50, 'VCCA',    'P');
#21=TERMINAL( T2,  3, #14, #52, #50, 'INPUTA',  'I');
#22=TERMINAL( T3,  4, #14, #53, #50, 'INPUTB',  'I');
#23=TERMINAL( T4,  7, #13, #54, #50, 'GNDA',    'G');
#24=TERMINAL( T5,  8, #11, #55, #50, 'GNDB',    'G');
#25=TERMINAL( T6, 11, #12, #56, #50, 'OUTPUTA', 'O');
#26=TERMINAL( T7, 12, #12, #57, #50, 'OUTPUTB', 'O');
#27=TERMINAL( T8, 14, #13, #58, #50, 'VCCB',    'P');

#30=DIE_SUBSTRATE_CONNECTION(.OPT., 'GNDA');

#40=SIZE_VALUE(1.312, 1.050);
#41=POSITION( 0, 0);
#42=POSITION(-0.0175,-0.042);
#43=POSITION(-0.042,-0.0175);
#44=POSITION(-0.042, 0.0175);
#45=POSITION(-0.0175, 0.042);
#46=POSITION( 0.0175, 0.042);
#47=POSITION( 0.042, 0.0175);
#48=POSITION( 0.042,-0.0175);
#49=POSITION( 0.0175,-0.042);
#50=ORIENTATION(0, .F., .F.);
#51=POSITION(-0.550, 0.416);
#52=POSITION(-0.502, 0.190);
#53=POSITION(-0.502,-0.192);
#54=POSITION(-0.399,-0.442);
#55=POSITION( 0.498,-0.442);
```

```
#56=POSITION( 0.511,-0.171);
#57=POSITION( 0.511, 0.171);
#58=POSITION( 0.558, 0.416);

#60=SIMULATOR_DATA('SPICE', 'SP7995.MOD', '1997-09-17', 'pSPICE',
  '4.0.1', '2G6');
#61=SIMULATOR_DATA('SPECTRE', 'SP7995.S', '1998-11-05', 'Spectre',
  '4.2.1 1992', '2G6, Level-3');

#70=FIDUCIAL_TYPE('fiduc1', '7995FID1.JIF', #72);
#71=FIDUCIAL(1, #70, #73, #74);
#72=SIZE_VALUE(0.072, 0.055);
#73=POSITION(-0.612, 0.470);
#74=ORIENTATION(0, .F., .F.);

ENDSEC;
END-ISO-10303-21;
```

Annex C
(informative)

Typical CAD view from the DDX file block example given in Annex A

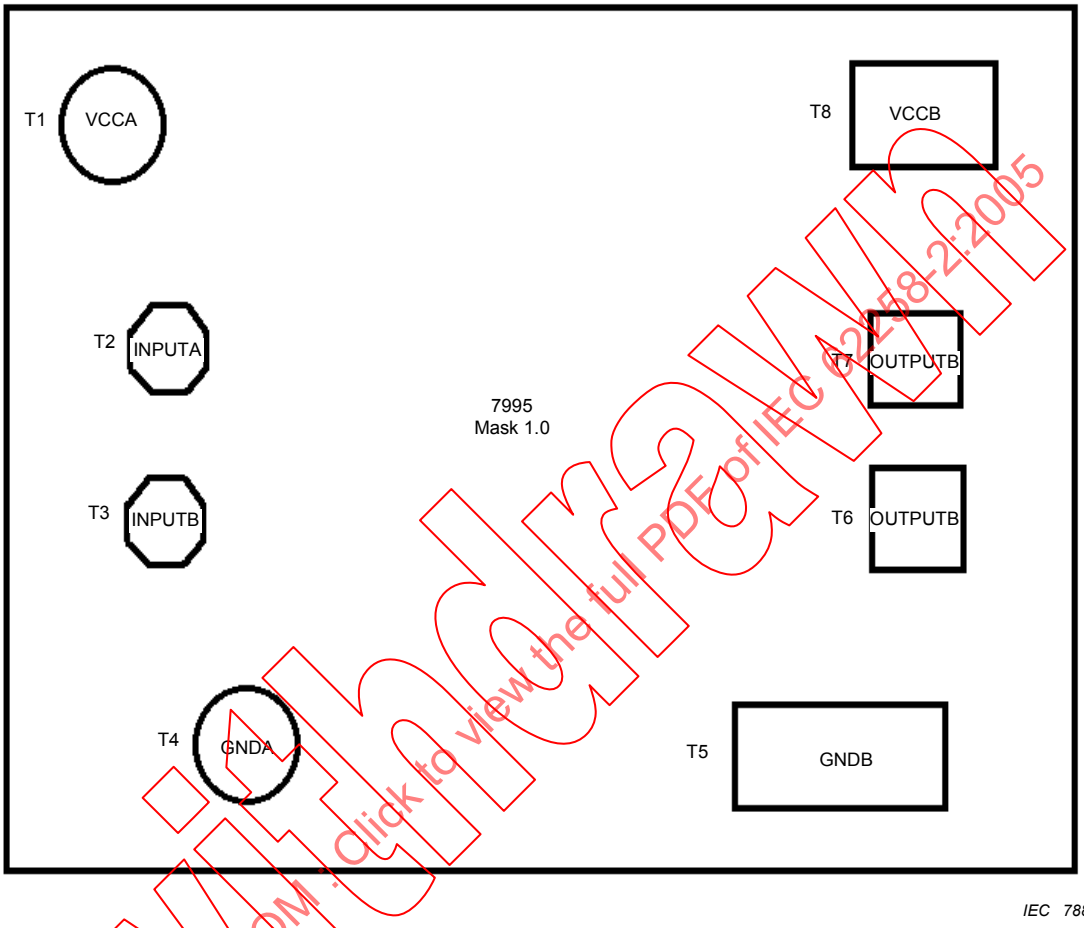

Figure C.1 – CAD representation of DDX example from Annex A

Note that Figure C.1 is not intended to be a geometrically accurate drawing, the purpose is solely to assist visualize the output from the DDX example given in Annex A. (The Die Fiducial graphic is not shown.) The CAD system has chosen to display the **DEVICE_NAME** (refer to 8.5) and the **DIE_MASK_REVISION** (refer to 8.6) parameters for the die. The individual terminal identifiers and terminal names have similarly been selected for display (refer to 8.24 for details).

Annex D
(informative)

Properties for simulation

A DDX file can reference other external files, such as simulation model files and thermal modelling files.

To do this adequately, the following data shall be present:

– the file name and its creation date,
– the exact name of the simulator,
– its version
– the relevant compliance level to which this model applies.

Within the model file there should be a textual note relating to the model's capability, applicability and accuracy, specifically noting any shortcomings of the model and its usage.

```
SIMULATOR_simulator_MODEL_FILE          text
SIMULATOR_simulator_MODEL_FILE_DATE     date
SIMULATOR_simulator_NAME                text
SIMULATOR_simulator_VERSION             text
SIMULATOR_simulator_COMPLIANCE          text  {e.g. 2G6, VHDL '93}
```

The text "*simulator*" shall be replaced with either the actual or generic name of the simulator used and/or model data. Typical *simulator* names might be Verilog, VHDL, SPICE, ELDO, BSDL, IBIS etc.,

All file names given shall not include any absolute computer drive, computer path reference, or any details that cause the information to become computer or operating system specific or dependent.

Example:

```
SIMULATOR_SPICE_MODEL_FILE = "RTBAE2.MOD";
SIMULATOR_SPICE_MODEL_FILE_DATE = "1997-09-17";
SIMULATOR_SPICE_NAME = "pSpice";
SIMULATOR_SPICE_VERSION = "4.0.1";
SIMULATOR_SPICE_COMPLIANCE = "2G6";
```